Demonstrating Scalability and Efficiency of Pack-Centric Resource Management for Cloud

Yi Wang, Ye Xia, Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA

Mauricio Tsugawa, Jose A. B. Fortes
Department of Electrical & Computer Engineering
University of Florida, Gainesville, FL 32611, USA

***Abstract*—Computational clouds have evolved to go beyond cost-effective on-demand hosting of IT resources and elasticity. The added features now include the ability to offer entire IT systems as a service that can quickly adapt to changing business environments. The new trend introduces challenges in datacenter resource management, including scalability, system-orientation, and optimization supporting both datacenter efficiency and customer system agility and performance. The conventional approach of resource management adopts a flat fine-grained model that results in problem formulations of enormous sizes; it also has the drawback of being less flexible in meeting customers' need. In this paper, we introduce a pack-centric approach to datacenter resource management by abstracting a system as a pack of resources and considering the mapping of these packs onto physical datacenter resource groups, called swads. The assignments of packs to swads are formulated as mixed integer programming problems. Scalability is achieved through a hierarchical decomposition method and parallel solvers. The new datacenter resource management framework is illustrated with a concrete resource placement problem. Numerical experiments show the scalability of the hierarchical decomposition method and the benefits of the overall framework.**

***Index Terms*—Cloud Computing, Datacenter, Virtual Machine Placement, Resource Management, Mixed Integer Programming**

I. Introduction

Computational clouds have become widely used by businesses as their most cost-effective means to deploy IT services. In order to serve a rapidly-growing customer population, datacenters have become increasingly large in scale and difficult to manage. Besides the scale, a fundamental development in the industry is the emergence of software-defined datacenters, which promise to offer complete virtualized infrastructure, including computing, network, storage, software, and all other datacenter resources, that can quickly adapt to changing business needs [1].

However, offering such sophisticated services poses a grave scalability challenge in datacenter resource management. We observe that virtually all existing management schemes are *VM-centric*, where each customer specifies a desired number of virtual machines (VM) as well as the resource requirements for each VM, including CPU, memory, storage, I/O throughput. The management system then assigns the VMs to the PMs (physical machines) such that certain cost, profit or performance objectives are optimized, subject to server resource constraints. Such a flat, fine-grained management scheme results in problem formulations of enormous sizes for large cloud systems (e.g., half a million PMs in Amazon EC2 [2]). The problem for determining the VM-to-PM assignment could involve billions of variables, and a great deal more than that if one considers traffic matrices and routing between VMs. The scalability challenge is further exacerbated by the complexity of the service offerings [1], [3]. For instance, an infrastructure/system deployment may have complex objectives and contain complex relationships among the system components, such as resource grouping and hierarchy, various colocation or anti-colocation constraints, topological relationships, workflow dependencies and traffic patterns.

This research was in part supported by the National Science Foundation under Grants 1265341 and 1234983 and by the National Science Foundation of China under Grant 61472256.

Existing solutions avoid the scalability challenge by restricting to smaller problems, limited supported features, and sub-optimal heuristics. For example, with simplifying assumptions, the problems may sometimes be reduced to multi-dimensional bin-packing [4]–[6], which is still NP-hard. Practical cloud systems usually adopt less sophisticated heuristics, such as round-robin, first-fit or first-fit-decrease, as evidenced by open-source middleware stacks [7]. While simple heuristics may find solutions quickly, they can also be underachieving with respect to the resource management objectives, such as resource utilization and/or workload performance.

A second challenge that the VM-centric approach faces is its lack of agility for group-based dynamic resource sharing. Customers may not know beforehand what exactly their resource requirements are for each of their VMs. The VM-centric resource allocation may lead to the situation where some VMs are not using their allocated resources while other VMs starve. Instead, the customer may wish to specify resource requirements in an aggregate manner and rely on dynamic sharing to avoid the aforementioned situation. Agility of such a high degree is not supported by today's cloud systems, although we can see signs of industry moving towards that direction. For instance, EC2's auto-scaling feature automatically increases or decreases the number of VMs based on the customer's workload [8]. But, this on-demand feature does not allow resource commitment to a group of VMs or in-group resource sharing.

To address these challenges, we propose a new *pack-centric* framework plus hierarchical decomposition for large-scale, sophisticated resource management. The new framework helps to break up an extremely complex, large resource management problem into a series of small, manageable ones based in large part on the natural grouping of VMs according to resource sharing and colocation requirements. Specifically, we introduce two concepts, called *pack* and *swad*; together with their hierarchies, they provide a recursively defined multi-level abstraction of customer demands and cloud resources. A collection of resource-sharing VMs is modeled as a pack, and multiple packs can be further abstracted into a higher-level pack, giving rise to a hierarchical organization of VMs and packs. Agility is achieved by pack-based resource management, allowing intra-pack resource sharing. Similarly, the resources in the cloud are organized into a multi-level hierarchical structure of PMs, i.e., swads of PMs, swads of swads, and so on. Datacenter resource management is transformed from a problem of VM-PM mapping to a problem of pack-swad mapping, with the problem size being progressively reduced as more levels of packs/swads are introduced.

The main goal of this paper is to demonstrate the scalability and effectiveness of the proposed pack-centric, hierarchical decomposition framework for datacenter resource management. The framework was first introduced in our work-in-progress paper [9]. We will outline the framework in this paper. One of its key components is the use of mixed integer programming (MIP) formulations and algorithms for resource management problems. In software-defined datacenters, the resource management problems are diverse and always changing, perhaps unknown ahead of the time [1], [3]. The MIP approach has the advantage of eliminating the need to craft specialized algorithms for different problems. For scalable solutions, we propose hierarchical decomposition of the problems in accordance with the pack and swad hierarchies. One of the main contributions of the paper is that we provide experimental evidence to demonstrate the high scalability of the hierarchical framework and the performance gain of the MIP approach. The paper is unique in that it evaluates MIP for solving large-scale datacenter resource management problems.

The idea of grouping the resource requirements from a customer and abstracting the corresponding hardware resources has been explored before. In v-Bundle [10], each VM has a minimum and a maximum resource requirement, and the VMs are initially assigned to the PMs based on their minimum requirements. The VMs form groups; the VMs in the same group can migrate among the assigned PMs for resource sharing. The distributed management scheme in v-Bundle does not provide optimal VM placement — both the initial VM placement and VM migration are ad-hoc (first-fit). In contrast, we take a completely different approach of hierarchical problem decomposition by a centralized controller, and we can achieve scalability and local optimality at the same time. Also, our pack abstraction and hierarchy allow the specification of resource sharing at multiple levels, instead of a single level in v-Bundle. Another prior work is the virtual datacenters (VDC) in [11]. A simple first-fit heuristic algorithm is used to select – for each customer – a server cluster (called VDC) that has sufficient resources to serve the customer's aggregate resource demand. Then a traditional VM-centric scheme is used for resource allocation within each VDC. Compared with v-Bundle and VDC, we have two clearly-defined hierarchies, a virtual hierarchy of requested resources and a physical hierarchy of available resources. Both are general, flexible and extensible beyond two levels. Our MIP-based formulations and algorithms are also general and flexible in that they can accommodate different cost criteria, customer requirements and constraints, and find high-performance solutions. Finally, MIP is used for assignments at all levels. Resource grouping and decomposition algorithms have also been proposed for several VM placement problems where the objective is to reduce datacenter network traffic or network energy cost [12]–[14].

II. Packs, Swads and Hierarchical Decomposition

A. Definitions and Examples of Pack and Swad

To meet the scalability and agility challenges, we propose a new pack-centric framework. The framework supports a variety of sophisticated, system-oriented cloud services. It also allows natural hierarchical grouping of resource requests, which can then be mapped to the hierarchy of physical resource groups in a datacenter. The architecture enables a scalable, hierarchical decomposition approach for solving large resource management problems.

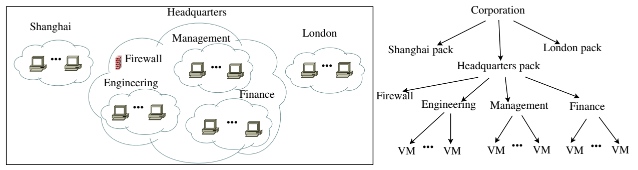

Fig. 1. VMs and other virtual resources can be organized through a hierarchical pack structure determined by administrative boundaries, locations and resource sharing requirement.

1) Packs and Pack Hierarchy: We propose a new abstraction called *pack*, which is a set of VMs, a set of smaller packs and/or collections of (virtual) resources that should be placed as a group in a datacenter for the purpose of resource sharing or performance enhancement. This recursive definition allows a customer to organize its resource requirement in a hierarchical structure, as illustrated by Fig. 1, which shows a scenario of a multinational corporation outsourcing its IT infrastructure to the cloud. The corporation has a branch in London, a branch in Shanghai, and its headquarters in San Jose, corresponding to three packs. The headquarters pack further consists of a firewall VM and three lower-level packs, describing the resource requirements by the management, finance, and engineering departments, respectively.

2) Swads and Swad Hierarchy: We define a *swad* as a set of PMs, collections physical resources (e.g., network storage) and/or lower-level swads in a cloud system. Each swad will be labeled with the types and amount of resources it possesses. The capacity of a resource in a swad is usually (but not always) equal to the sum of the capacities of its components, possibly excluding a certain percentage of resources that may be set aside to support elasticity. In other times, more detailed description of a resource in a swad is needed, such as the networking capacity of a server cluster when the cluster is viewed as a swad.

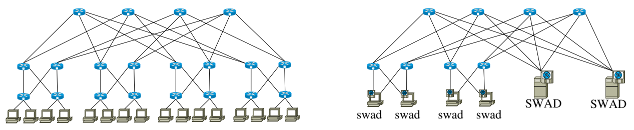

Fig. 2. Swad-based hierarchical abstraction of a cloud system

In the example of Fig. 2 with a fat-tree topology [15], we use a swad to represent the PMs of each rack, shown by the nodes labeled with "swad" in the right plot. We then group a number of lower-level swads into a higher-level swad, as illustrated by the nodes labeled with "SWAD", giving rise to a hierarchical structure (tree), with the whole datacenter as a swad at the root. While the fat-tree topology in Fig. 2 has two levels (pod and rack), the swad hierarchy may have an arbitrary number of levels.

3) Construction of the Hierarchies: The swad hierarchy is established by the cloud provider alone according its resource management policies, proximity of physical resources and various other constraints. The construction of the swad hierarchy is expected to be straightforward in most cases. For ease of design and maintenance, large datacenters usually contain zones of identical hardware and software systems. Within each such zone, the swads can be made identical, each consisting of identical system components.

The pack hierarchy is established by combining the customers' pack specifications and the cloud provider's considerations. A customer's request may be already in the form of a pack hierarchy, such as the example of Fig. 1. Each customer's pack hierarchy will be a subtree in the final pack hierarchy. The provider can group multiple pack hierarchies from the customers into a higher-level pack. The grouping process continues upward recursively to form the final pack hierarchy. The pack hierarchy is constructed after the swad hierarchy has been established so that the former can "fit" the latter. The pack hierarchy can be modified frequently and substantially in response to customers' ever-changing pack requests. Unlike the swad hierarchy, there is a great deal of flexibility in constructing the pack hierarchy. A thorough exploration is beyond the scope of this paper.

B. Hierarchical Decomposition and Pack-to-Swad Assignment

Aside from the benefits that packs allow system-oriented cloud services and increased customer agility, the other main use of the pack and swad hierarchies is to decompose a large resource management problem of enormous complexity into a series of much smaller subproblems that are far easier to solve quickly and can be solved in parallel.

The hierarchical decomposition algorithm is shown in Algorithm 1. Consider an arbitrary pack hierarchy (or sub-hierarchy) rooted at pack p^* and an corresponding swad hierarchy (or sub-hierarchy) rooted at swad s^*. Let P and S denote the sets of the direct child packs and child swads of p^* and s^*, respectively. The algorithm first assigns the packs in P to the swads in S (Line 2). The assignment is done by solving an MIP problem (see Sections III and IV). The assignment process then continues downward along the swad hierarchy (Lines 4-20). In general, consider a swad at level i. Assume that some level-i packs have been assigned to that swad. For that swad, the next assignment to do is to assign the child packs of the those level-i packs to the child swads of the level-i swad (Line 17). When the assignment process finishes, all the VMs will be assigned to the PMs.

There is an assignment subproblem associated with each swad on the swad hierarchy. As the experimental results in Section IV show, the computation time for the subproblems at the lowest levels dominates the overall computation time. These subproblems are independent from each other, and therefore, can be dispatched to separate solvers for parallel computation.

Several things may happen. In any assignment step above, if a pack is too large to fit into any child swad of the swad currently being considered, then the pack will be unpacked and replaced by its children (Line 1). Such a unpacking process can continue until either every pack can fit into some child swads, or until there are no more descendant packs to unpack. If a terminal swad (whose children are PMs) is reached but some of the packs assigned to the swad have descendant packs, all the descendant packs will be unpacked and made ready for the final assignment step.

III. A Canonical VM Placement Problem

To demonstrate the effectiveness and scalability of our framework, we will use a canonical datacenter resource management problem as an example. In this section, we will describe the MIP formulation of the problem and the application of hierarchical decomposition (Algorithm 1) to this problem. Later in Section IV, we will solve very large instances of the problem. Although many other problems can be chosen for demonstration, the canonical problem has the advantage of wider applicability and ease of presentation.

Algorithm 1 Hierarchical Decomposition Algorithm (P, S)

```
1: check for possible unpacking conditions and unpack;
2: find an assignment of packs in $P$ to swads in $S$ by solving an MIP problem;
3: if an optimal assignment $X = [x_{ik}]_{\forall p \in P, s \in S}$ is found then
4:     for each swad $s \in S$ do
5:         if $s$ is a PM then
6:             continue;                                  ▷ assignment to $s$ is done
7:         else                                           ▷ $s$ is a swad with children
8:             $P' \leftarrow \emptyset$, $S' \leftarrow s.children$;
9:             for each pack $p \in P$ with $x_{ps} = 1$ do
10:                if $p$ is a VM then
11:                    insert $p$ into $P'$;
12:                else                                   ▷ $p$ is a pack with children
13:                    insert $p.children$ into $P'$;
14:                end if
15:            end for
16:            if $P' \neq \emptyset$ then
17:                execute Algorithm 1 with $(P', S')$;
18:            end if
19:        end if
20:    end for
21: end if
```

A. Problem Formulation

Consider a problem of assigning N VM to M PMs in a datacenter, under the capacity constraints of the physical resources of the PMs. Specifically, we consider three types of physical resources: the number of vCPU, memory size (GB) and local disk size (GB). Each VM requires certain amount of resources in these three categories. The total requested resources by the VMs assigned to a PM must not exceed the PM's resource capacities.

The optimization objective is to minimize the total operation cost of running the active PMs. We assume that a fixed operation cost is incurred for a PM as long as the PM is used by some VMs, i.e., some VMs are assigned to the PM. When a PM has no VMs assigned to it, it will be turned off, and there is zero operation cost. The operation cost may include the average energy cost when a machine is running and typical maintenance cost.

Let the set of VMs and PMs be denoted by $\mathcal{V} = \{1, 2, \ldots, N\}$ and $\mathcal{P} = \{1, 2, \ldots, M\}$, respectively. Eeach VM $i \in \mathcal{V}$ requires a_i vCPUs, b_i GB memory, and d_i GB local disk size. For each PM $k \in \mathcal{P}$, its resource capacities are A_k vCPUs, B_k GB of memory and D_k GB of local disk size.

For each VM $i \in \mathcal{V}$ and each PM $k \in \mathcal{P}$, let x_{ik} be a 0-1 assignment variable from VM i to PM k, which takes the value 1 if VM i is assigned to PM k and 0 otherwise. Each VM i should be assigned to only one PM.

For each PM k, let y_k be a 0-1 variable indicating whether PM k is used by some VMs, with $y_k = 1$ meaning that PM k is used (active) and $y_k = 0$ meaning that PM k is unused (which will be turned off). When PM k is used to serve some VMs, there is a fixed cost C_k associated with running the PM.

The following is the MIP formulation of the problem. The PM resource capacity constraints associated with the vCPU, memory and disk space, respectively, can be expressed as: $\sum_{i \in \mathcal{V}} a_i\, x_{ik} \leq A_k$, $\sum_{i \in \mathcal{V}} b_i\, x_{ik} \leq B_k$ and $\sum_{i \in \mathcal{V}} d_i\, x_{ik} \leq D_k$, $\forall k \in \mathcal{P}$. To ensure that each VM is assigned to exactly one PM, we need the constraints: $\sum_{k \in \mathcal{P}} x_{ik} = 1, \forall i \in \mathcal{V}$. To ensure that a PM k must be set active (i.e., $y_k = 1$) as long as some VMs are assigned to it, we need: $\sum_{i \in \mathcal{V}} x_{ik} \leq N\, y_k, \forall k \in \mathcal{P}$. Finally, the optimization objective is: $\min \sum_{k \in \mathcal{P}} C_k\, y_k$.

B. Algorithm Implementation

In our experiments, customers make requests of VMs and/or packs of VMs (see Section III-C). We implement three algorithms for the problem in Section III-A: (1) the VM-centric flat MIP that directly assigns all the VMs to all the PMs; (2) the proposed hierarchical decomposition algorithm (Algorithm 1); (3) a heuristic algorithm improved from the first-fit-decreasing (FFD) algorithm for bin-packing problems.

1) VM-Centric Flat MIP: All the packs are unpacked into separate VMs. The swad hierarchy is flattened into separate PMs. Then, the VM-to-PM assignment problem in Section III-A is solved directly using an MIP solver.

2) Pack-Centric Hierarchical Decomposition: For the purpose of demonstrating scalability, we implement a two-level version of the hierarchical decomposition algorithm. We will show that two levels are enough for fairly large instances of the problem at hand. In practice, the problems are different for different datacenters and some problems may require more than two levels to achieve sufficient scalability. There are also other reasons to have more than two levels, such as satisfying the resource-sharing requirements from the customers.

All VM/pack requests are re-grouped into multiple height-1 packs; this is done randomly except that each of the requested packs is kept together when possible. The PMs in the datacenter are grouped into multiple height-1 swads, each with a mix of different PM types. For these two-level pack/swad hierarchies, two levels of MIP are needed. The first level (the top level) has one MIP subproblem to solve, which is to assign the packs to the swads. The second level (the bottom level) has as many MIP subproblems as the number of swads used (i.e., the swads with assigned packs), one for every used swad. For each of the second-level subproblems, all the packs assigned to the focal swad are unpacked into VMs. Then, the VMs are assigned to the PMs in the swad by solving a smaller MIP problem similar to the one in Section III-A.

3) Heuristic First-Fit-Decreasing Algorithm: For comparison, we also provide a heuristic algorithm specifically tailored for the problem in Section III-A. It is derived from and improves upon the first-fit-decreasing (FFD) algorithm for bin-packing. In our improved FFD heuristic, the VMs are sorted in decreasing order of their monetary costs and the PMs are sorted in increasing order of their operation costs . Then, for each VM in the sorted VM list, we assign the VM to the first suitable PM (first-fit) in the sorted PM list. The reason we use costs for sorting is that they generally correspond well with the resource requirements of VMs and the capabilities of PMs, as can be seen from Table I . The heuristic algorithm adopts the traditional VM-centric view. Its time complexity is $O(NM)$.

C. Experimental Environment

The VM types and PM types used in the experiments are specified in Table I, which are from Amazon's EC2 [8]. The operation costs in 5th column of PM types table are our estimates based on the PM configurations, since the actual costs are unavailable. The numbers are normalized with the least powerful PM type s1 having an operation cost 100.

To simulate pack-centric scenarios, we allow two types of requests in each test case: individual VM requests and pack requests. Individual VM requests refer to individual customers requesting a single VM or a few independent VMs, which can be freely placed on any PM in the datacenter. A pack request is a bundle of VMs requested by an enterprise customer. A pack may have certain internal structures and relationship among the VMs. For instance, they may communicate with each other or depend on each other in a workflow; thus it is preferable to place them in the same swad. We assume in our experiments that each pack request can fit into at least one of the swads in the datacenter. In real world situations with very large pack requests, the datacenter can aggregate swads into higher-level swads to accommodate large packs.

TABLE I
AMAZON EC2 VM TYPES AND PM TYPES

VM Type	vCPU	Memory (GB)	Storage (GB)	Cost ($\$/hour$)
m3.medium	1	3.75	1 × 4	67
m3.large	2	7.5	1 × 32	133
m3.xlarge	4	15	2 × 40	266
m3.2xlarge	8	30	2 × 80	532
c3.large	2	3.75	2 × 16	105
c3.xlarge	4	7.5	2 × 40	210
c3.2xlarge	8	15	2 × 80	420
c3.4xlarge	16	30	2 × 160	840
c3.8xlarge	32	60	2 × 320	1680
r3.large	2	15.25	1 × 32	175
r3.xlarge	4	30.5	1 × 80	350
r3.2xlarge	8	61	1 × 160	700
r3.4xlarge	16	122	1 × 320	1400
r3.8xlarge	32	244	2 × 320	2800
i2.xlarge	4	30.5	1 × 800	853
i2.2xlarge	8	61	2 × 800	1705
i2.4xlarge	16	122	4 × 800	3410
i2.8xlarge	32	244	8 × 800	6820
PM Type	vCPU	Memory (GB)	Storage (GB)	Cost (normalized)
s1	8	16	1 × 256	100
s2	8	32	1 × 512	150
s3	8	64	2 × 512	200
s4	8	64	4 × 512	250
m1	16	32	2 × 512	300
m2	16	64	4 × 512	400
m3	16	128	4 × 1000	500
m4	16	256	8 × 1000	700
m5	16	256	16 × 512	700
l1	32	256	4 × 1000	800
l2	48	512	8 × 1000	1200
l3	64	1024	4 × 1000	1500
l4	80	2048	16 × 1600	2200
l5	120	4096	4 × 1000	2500
l6	120	4096	24 × 1600	3000

In each experiment, the input data contains a mixture of individual VM requests and pack requests of different VM types. Each swad is a heterogeneous group of PMs of different types. The datacenter's swad hierarchy contains 1 to 250 different swads, depending on the test cases. From Test 1 to Test 10, the numbers of VMs and PMs increase, and the proportions of different VM or PM types also vary.

We use Gurobi Optimizer 6.5 [16] as the MIP solver. All the experiments are conducted on a PC with AMD quad-core 3.5 GHz CPU and 16 GB memory. We collect the running time and the achieved objective values as the performance metrics.

IV. Experimental Results

A. Solving VM-Centric Flat MIP

The VM-centric flat MIP yields optimal solutions, which provide useful performance benchmarks. However, only small problem instances can be solved due to its lack of scalability. Our experimental results confirm this. Fig. 3 shows the computation time grows extremely fast from Test 1 to Test 7 when the numbers of VMs/PMs increase from 90 VMs/70 PMs to 600 VMs/375 PMs. The computation time reaches 2070 seconds in Test 7. Any instance with more than 1000 VMs and 1000 PMs cannot be solved optimally in reasonable time on our machine.

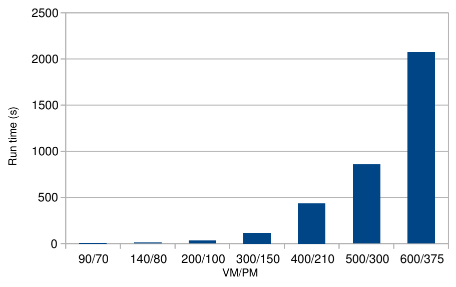

Fig. 3. Computation time of VM-centric flat MIP

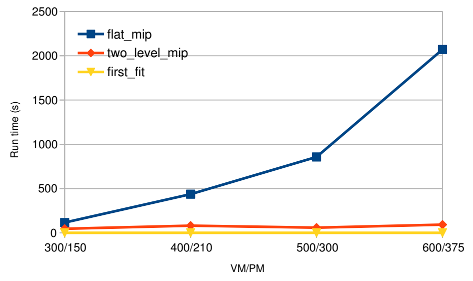

Fig. 5. Comparison of computation time of the algorithms

B. Comparing Three Algorithms

In this part of the experiments, we use test cases 4 - 7, each of which has more than one swad but also is small enough to be solved by the flat MIP. The three algorithms from Section III-B: the flat MIP, the hierarchical decomposition algorithm (Algorithm 1) with two-level hierarchies, and the FFD heuristic, are compared for each test case.

TABLE II
COMPARISON OF ALGORITHMS

Test No.	4	5	6	7
No. of VMs	300	400	500	600
No. of PMs	150	210	300	375
No. of Swads	2	3	4	5
Flat MIP Cost	46300	48800	60800	70300
Run Time	115.88	436.87	856.99	2070.67
Two-level HD Cost	46600	49050	64900	76000
Total Run Time	45.80	80.99	58.46	92.34
FFD Cost	65800	78500	100000	119500
Run Time	0.026	0.041	0.050	0.10

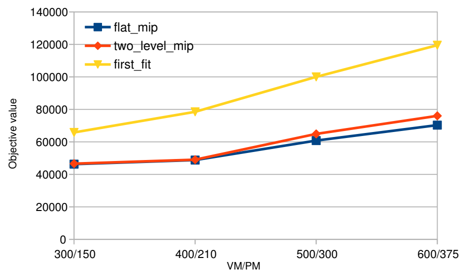

Fig. 4. Comparison of the objective values of the algorithms

In the two-level hierarchical decomposition algorithm, the datacenter has 2, 3, 4, 5 height-1 swads in Test 4, 5, 6, 7, respectively. Each swad contains 70 to 75 PMs of different types. As shown in Algorithm 1, the overall problem needs two levels of MIP to solve. In the top level pack-to-swad assignment, the total resource requirements of each pack and the total resource capacities of each swad are used as the input parameters for an MIP problem. The objective of that problem is to minimize the total cost of all the swads used. We define a safety margin $0 < \beta < 1$ for resource usage. The constraints are that each swad can have at most a fraction β of its total capacity of each resource type to be used in the first-level pack-to-swad assignment. The purpose of having $\beta < 1$ is to reserve more room for maneuver during the second-level VM-to-PM assignment within each swad. If we leave no margin, i.e. $\beta = 1$, a swad might be packed too tightly with VMs/packs and the second-level assignment might fail due to the heterogeneity of the VMs and PMs. We used $\beta = 0.7$ in our experiments. Each second-level VM-to-PM assignment does not have a safety margin β.

The experimental results are shown in Table II, Fig. 4, and Fig. 5. In the table, 'HD' stands for 'hierarchical decomposition'. The advantages of the hierarchical decomposition framework can be observed from the computation time and achieved objective values in the results. First, it greatly reduces the computation time compared with the flat MIP approach. The computation time of the hierarchical decomposition algorithm increases slowly as the problem size increases, and hence, the speed gain against the flat MIP gets much larger. In Test 7, the computation time of the hierarchical decomposition algorithm is only 92.34 seconds, whereas it is 2070.67 seconds for the flat MIP. The improvement is due to the fact that the overall problem is decomposed into a series of smaller subproblems, which can be solved much more quickly. Note that, in our experiments, the subproblems are solved in sequence on the same PC. The reported computation time here is the total time. Since the bottom-level MIP subproblems are completely independent from each other, in practice, they can be distributed to the swads and solved locally in parallel. Thus, the overall run time is much shorter. We will explore such scalability properties in more details in Section IV-C.

The second benefit of the proposed framework is that the achieved objective values are very close to the optimal values, only 4.0% larger than the latter on average (the optimal values are obtained by the flat MIP). The near optimal results are consistent across the four test cases, with the largest difference being 8.1%. Together with the great gain in computation speed, the hierarchical decomposition algorithm compares favorably to the flat MIP approach, especially for larger-scale datacenter resource management problems.

The heuristic FFD algorithm is the fastest on these small test cases, which is expected due to its $O(NM)$ complexity. However, its achieved objective values are on average 59.4% larger than the optimal values, which is a fairly large cost increase. The poorer performance can be partly explained by the heterogeneity of the swads and the PMs inside a swad, which makes the FFD scheme less suitable in many situations.

C. Scalability of Hierarchical Decomposition Algorithm

In this part of the experiments, we focus on exploring the scalability of the hierarchical decomposition algorithm. The test cases 8, 9, and 10 each have 6000, 20000 and 50000 VMs with different mixes of individual VM requests and pack requests of all VM types. The datacenter has 30, 100, 250 swads for the test cases 8, 9 and 10, respectively, with each swad having 100 PMs of mixed types. The large cases, such as 50000 VMs and 25000 PMs, correspond well with the scale of current datacenters of large cloud providers.

TABLE III
ALGORITHM SCALABILITY

Test No.	8	9	10
No. of VMs	6000	20000	50000
No. of PMs	3000	10000	25000
No. of Packs	105	358	754
No. of Swads	30	100	250
HD 1st-Level Time	0.065	6.40	3.98
2nd-Level Average Time	52.54	50.99	76.99
Total Cost	758250	2647900	5572850
Total Time	1377.87	4637.03	14672.82
FFD Cost	1085400	3704800	8260600
Run Time	10.21	114.54	684.32

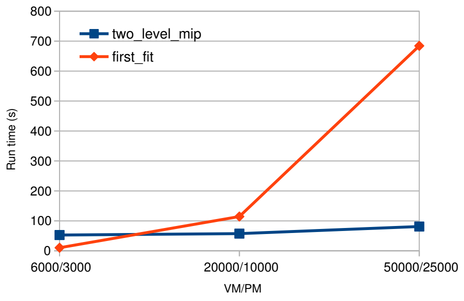

Fig. 6. Algorithm scalability

For each test case, the two-level hierarchical decomposition algorithm is compared only with the heuristic FFD algorithm, since the flat MIP cannot solve such large problems. The results are shown in Table III and Fig. 6. For the hierarchical decomposition algorithm, we show the computation time for the top level (1st-level) pack-to-swad assignment, the average computation time for the bottom-level (2nd-level) VM-to-PM assignment in each of the swads, the total operating cost after all assignments are made, and the total algorithm running time. Our experiments use only one PC to run all the bottom-level assignments in sequence. If each swad has a separate controller to solve its own VM-to-PM assignment subproblem, the total computation time from an individual swad's point of view is the time to solve the top-level assignment problem plus the time to solve one bottom-level subproblem.

When the number of VMs increases from 6000 to 50000 and the number of PMs increases from 3000 to 25000 (from test 8 to test 10), the average computation time faced by a swad, including that for both levels, only sees a modest increase from 53.19 seconds to 80.97 seconds. That is because the computation time is dominated by the bottom-level computation time, and by keeping the swad size to be a manageable 100 PMs, the bottom-level computation time remains within 30 to 100 seconds from different swads, and is relatively predictable. Scaling the problem size even larger will only increase the top-level computation time, if the bottom-level subproblems are solved by parallel servers. As the problem size increases further, the levels of decomposition can be increased (if needed) so that the pack-to-swad assignment can also be solved in parallel. For our problem, we see that even two-level decomposition is sufficient for problem sizes required by today's datacenters.

The heuristic FFD algorithm, on the other hand, scales linearly with $M \times N$. Though it is still relatively fast, it does suffer from dramatic slow-down as the problem size becomes large. Starting from test case 9, the computation time exceeds that of the hierarchical decomposition algorithm. Moreover, the achieved objective value by the FFD algorithm is on average 43.8% more than that achieved by the hierarchical decomposition algorithm, which is another disadvantage of the heuristic algorithm. The heuristic FFD algorithm can be parallelized by splitting the VMs and PMs into smaller batches and having different controllers to run FFD on different pairs of VM-PM batches. However, the achieved objective value will get worse.

V. CONCLUSIONS

In this paper, we outline a pack-centric hierarchical decomposition framework for datacenter resource management, as opposed to the traditional, flat, VM-centric approach adopted by much prior work. Our experiments have shown that the new framework is very effective and highly scalable, capable of solving very large problems encountered by large datacenters and leading to much reduced datacenter costs. By relying on MIP, the framework is well suited for describing and solving other sophisticated, system-oriented resource management problems. It can enhance customer agility, while at the same time achieve high resource efficiency in datacenters.

REFERENCES

[1] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software defined infrastructures," *IBM Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.
[2] H. Liu, *Amazon Data Center Size*, March 2012, http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/.
[3] W. C. Arnold, D. J. Arroyo, W. Segmuller, M. Spreitzer, M. Steinder, and A. N. Tantawi, "Workload orchestration and optimization for software defined environments," *IBM Journal of Research and Development*, vol. 58, no. 2/3, March/May 2014.
[4] M. Chen, H. Zhang, Y. Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," in *Proceedings of IFIP/IEEE Integrated Network Management (IM)*, 2011.
[5] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '14)*, 2014.
[6] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2647–2660, 2014.
[7] *OpenStack Project*, 2015, http://www.openstack.org/.
[8] *Amazon AWS EC2 Instances*, 2015, http://aws.amazon.com/ec2/.
[9] Y. Xia, M. Tsugawa, J. Fortes, and S. Chen, "Hierarchical mixed integer programming for pack-to-swad placement in datacenters (work in progress)," *Proceedings of IEEE International Conference on Autonomic Computing (ICAC)*, 2015.
[10] L. Hu, K. D. Ryu, D. D. Silva, and K. Schwan, "v-Bundle: Flexible group resource offerings in clouds," in *IEEE International Conference on Distributed Computing Systems (ICDCS '12)*, June 2012.
[11] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," *Proceedings of ACM CoNEXT*, 2010.
[12] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of IEEE INFOCOM*, 2010.
[13] M.-T. Chen, C.-C. Hsu, M.-S. Kuo, Y.-J. Cheng, and C.-F. Chou, "GreenGlue: Power optimization for data centers through resource-guaranteed VM placement," in *IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom), and Cyber, Physical and Social Computing(CPSCom)*, 2014.
[14] L. Zhang, X. Yin, Z. Li, and C. Wu, "Hierarchical virtual machine placement in modular data centers," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2015.
[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Proc. of ACM SIGCOMM*, 2008.
[16] *Gurobi Optimizer*, 2015, http://www.gurobi.com.